

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

EV355229215

**DV Metadata Extraction**

**Inventors:**

Charles Alan Ludwig

James Dooley

ATTORNEY'S DOCKET NO. MS1-1708US

1  
2 **TECHNICAL FIELD**

3       The present disclosure generally relates to processing multimedia data, and  
4 more particularly, to extracting metadata from DV formatted multimedia data.  
5

6 **BACKGROUND**

7       DV is a digital video format used world-wide for digital video cameras.  
8 The DV format is an international standard that was created by a consortium of  
9 companies typically referred to as the DV consortium. DV, originally known as  
10 DVC (Digital Video Cassette), uses a metal evaporate tape to record very high  
11 quality digital video. The DV video specification, IEC 61834, specifies the  
12 content, format and recording method of data blocks forming helical records on the  
13 digital tape. It also describes the common specifications for cassettes, modulation  
14 method, magnetization and basic system data, for digital video cassette recording  
15 systems using 6.35 mm (1/4 inch) magnetic tape, and the electrical and mechanical  
16 characteristics of equipment which provides for the interchangeability of recorded  
17 cassettes.

18       DV video information is carried in a data stream at a rate of about 29  
19 megabits per second (3.7 MByte/sec). A DV video frame typically includes 10  
20 DIF sequences, each of which consists of 150 DIF blocks having 80 bytes of data  
21 each. In addition to video and audio data, each DV video frame includes extra  
22 data associated with the video and audio data called DV metadata.

23       DV metadata can include a wide range of data associated with the video  
24 data in a DV frame. For example, DV metadata can include the time and date that  
25 video was recorded, various settings on the camcorder at the time the video was

1 recorded, and so on. According to IEC 61834-4, DV metadata is divided into 256  
2 separate "packs". Although 256 packs are reserved for DV metadata, many of the  
3 packs have yet to be defined. Each pack consists of 5 bytes. The first byte in each  
4 DV metadata pack is the pack ID, and the next four bytes consist of binary fields.

5 The DV format permits each DV video frame to stand on its own without  
6 having to rely on any data from preceding or following frames. For example, the  
7 same metadata is repeated numerous times within a DV frame. The redundancy  
8 built into each DV frame and the wealth of additional data (i.e., metadata)  
9 inherent to the DV format make DV video an ideal format for editing. However,  
10 various difficulties prevent current DV editing applications from taking full  
11 advantage of the unique features of DV video that make it ideally suited for  
12 editing.

13 In a typical DV scenario, video is recorded and converted to digital form in  
14 a camcorder. The video data on the digital tape can be played in a digital tape  
15 drive, such as the one in the camcorder, in a DVCR, or in a standalone unit. DV  
16 data can be transferred electronically via firewire to a computer's hard disk. The  
17 transfer process is typically performed by a capture driver, a standalone utility, or a  
18 component of an editing application executing on a computer such as a desktop  
19 personal computer. During the transfer process, the DV data is "wrapped" into a  
20 file format commonly understood by computers, such as AVI for Windows or  
21 Quicktime for the Mac. Therefore, once the transfer process is finished, the DV  
22 data on the computer hard drive is wrapped in a file format that standard editing  
23 applications can process. Various editing applications, such as Adobe®  
24 Premiere® Pro, enable nonlinear video editing through real-time video and audio  
25 editing tools.

1           However, as indicated above, current DV editing applications take little or  
2 no advantage of the rich information provided in the DV video format that makes  
3 it ideally suited for editing. The main reason for this is that it is difficult to extract  
4 the DV metadata from within DV video frames. DV metadata extraction currently  
5 requires an application developer to write its own custom code for extracting  
6 specifically desired metadata. In addition, an extraction process implemented by a  
7 DV editing application would be very processor intensive, which would hinder the  
8 performance of other editing functions of the application. One consequence of  
9 these difficulties is that DV metadata is generally not exploited by most DV  
10 editing applications.

11           Accordingly, a need exists for a way to extract DV metadata from DV data  
12 streams.

## 13 14 **SUMMARY**

15           Extraction of DV metadata from a DV data stream is described herein.

16           In accordance with one implementation, an instruction is received  
17 specifying additional per-frame DV metadata to be extracted from a DV data  
18 stream. The metadata is extracted from DV frames of the DV data stream.

19           In accordance with another implementation, metadata is stored in a  
20 container and the container is attached to a DV sample of a DV frame. The  
21 container is manageable to have additional DV metadata structures stored within it  
22 and to provide for the retrieval of metadata items that have been stored within it.

23           In accordance with yet another implementation, a DV metadata structure is  
24 stored within the container. The DV metadata structure is an unpacked version of  
25 a DV metadata pack. The DV metadata structure includes binary values unpacked

1 from the DV metadata pack and a different variable name associated with each  
2 binary value.

### 3 4 **BRIEF DESCRIPTION OF THE DRAWINGS**

5 The same reference numerals are used throughout the drawings to reference  
6 like components and features.

7 Fig. 1 illustrates an exemplary environment suitable for DV metadata  
8 extraction.

9 Fig. 2 illustrates an exemplary embodiment of a computer suitable for  
10 extracting DV metadata from a DV data stream.

11 Fig. 3 illustrates an example of DV data being processed through various  
12 processing components.

13 Fig. 4 illustrates components of a DV metadata extraction tool:

14 Fig. 5 illustrates a container having one or more DV metadata packs and  
15 one or more unpacked DV\_METADATA structures.

16 Fig. 6 illustrates a block diagram of an exemplary method for extracting DV  
17 metadata from a DV data stream.

18 Fig. 7 illustrates a block diagram of another exemplary method for  
19 extracting DV metadata from a DV data stream.

20 Fig. 8 illustrates an exemplary computing environment suitable for  
21 implementing a computer such as that in Fig. 2.

## DETAILED DESCRIPTION

### Overview

The following discussion is directed to a set of interfaces, data structures and events for representing a DV metadata extraction tool. The DV metadata extraction tool includes an API (application programming interface) for describing DV metadata packs that are to be extracted from a DV data stream. The extraction API is called IMFExtractDVMetadata, and it supports methods for specifying and removing DV metadata packs to be extracted from DV frames. The extraction API also supports methods for determining the number of DV metadata packs in an extraction list and determining the ID of a DV metadata pack at a given index in the extraction list.

The DV metadata extraction tool also includes an API for describing a container that holds DV metadata once it has been extracted from a DV frame. The container API is called IMFDVMetadataContainer, and it supports methods for adding and removing DV structures to the container and for retrieving data from the container, removing data from the container, and iterating through the container. The DV metadata extraction tool also includes a collection of higher-level structures that represent unpacked DV metadata packs.

The DV metadata extraction tool is generally described within the context of the Media Foundation architecture by Microsoft® Corporation. However, it is noted that the DV metadata extraction tool is designed and described herein in a manner that enables its use in any suitable multimedia architecture.

## Exemplary Environment

Fig. 1 illustrates an exemplary environment 100 that is suitable for DV metadata extraction. The exemplary environment 100 of Fig. 1 includes a computer 102 and one or more DV video input sources 104.

DV video input sources 104 can be any type of device or communication network capable of transferring DV video content to computer 102, including for example, portable storage media 104(1) (e.g., magnetic discs, media cards, optical discs), a DV video recording device 104(2) (e.g., a digital camcorder), or a network 104(3) such as the Internet, a corporate network, or a home network.

Video recording device 104(2) can be any of various digital recording devices capable of recording live-motion video and audio in DV format (i.e., on a digital tape) for later replay via a digital tape drive, for example, in a DVCR, a camcorder, or a personal computer such as computer 102. A video recording device 104(2) is typically capable of being connected directly to computer 102 using an i.LINK (IEEE 1394) or FireWire digital interface, so that DV video content can be edited directly on the computer 102.

Computer 102 may be implemented as various computing devices generally capable of receiving video content from various sources 104 and manipulating the video content for editing and playback through a resident multimedia architecture such as the Media Foundation architecture by Microsoft Corporation, for example. Computer 102 is otherwise typically capable of performing common computing functions, such as email, calendaring, task organization, word processing, Web browsing, and so on. In the described embodiments, computer 102 runs an open platform operating system, such as the Windows<sup>®</sup> brand operating systems from Microsoft<sup>®</sup>. Computer 102 may be implemented, for example, as a desktop

1 computer, a server computer, a laptop computer, or other form of personal  
2 computer (PC). One exemplary implementation of computer 102 is described in  
3 more detail below with reference to Fig. 8.

4 As discussed in greater detail below with reference to the exemplary  
5 embodiments, computer 102 is generally configured with a multimedia architecture  
6 that includes a DV metadata extraction tool enabling the extraction of DV  
7 metadata from DV data.

### 8 9 **Exemplary Embodiments**

10 Fig. 2 illustrates an exemplary embodiment of a computer 102 suitable for  
11 extracting DV metadata from a DV data stream. A multimedia architecture and  
12 related components facilitating DV metadata extraction are described throughout  
13 this disclosure in the general context of computer/processor-executable  
14 instructions, such as program modules being executed by a personal computer.  
15 Generally, program modules include routines, programs, objects, components, data  
16 structures, etc., that perform particular tasks or implement particular abstract data  
17 types. Moreover, those skilled in the art will appreciate that such program  
18 modules may be implemented using other computer system configurations,  
19 including hand-held devices, multi-processor systems, microprocessor based or  
20 programmable consumer electronics, network PCs, minicomputers, mainframe  
21 computers, and the like. Furthermore, such program modules may also be  
22 practiced in distributed computing environments where tasks are performed by  
23 remote processing devices that are linked through a communications network. In a  
24 distributed computing environment, program modules may be located in both local  
25 and remote memory storage devices. In the current computing environment of Fig.



1 2, computer 102 is generally illustrated as having program modules located in a  
2 local memory (not shown). As indicated above, an exemplary implementation of  
3 computer 102 is described in greater detail below with reference to Fig. 8.

4 The DV metadata extraction tool 200 shown in Fig. 2 may operate in the  
5 context of a multimedia architecture 202 such as Microsoft's Media Foundation.  
6 However, the DV metadata extraction tool 200 is not limited to operation in such  
7 an architecture 202. Thus, the DV metadata extraction tool 200 might also be  
8 implemented, for example, as a stand alone component or a subcomponent of  
9 another application program. Prior to describing the DV metadata extraction tool  
10 200, a brief description of the multimedia architecture 202 will be provided with  
11 reference to Fig. 2.

12 As shown in Fig. 2, multimedia architecture 202 includes various  
13 component layers: In addition, multimedia architecture 202 also generally includes  
14 supporting or associated media applications 204. Such applications 204 are  
15 illustrated in Fig. 2 separately from the multimedia architecture 202, but might also  
16 be shown as a part of the architecture 202. The component layers of multimedia  
17 architecture 202 include control component layer 206, core component layer 208,  
18 base component layer 210, development platform layer 212 and definition layer  
19 214.

20 Components of control component layer 208 include media processor 234,  
21 basic editor 218, basic encoder 220, media session 222, topology loader 224,  
22 media engine 226, and source resolver 228. These components generally make up  
23 task oriented API's (application programming interfaces) that may be fully  
24 managed or un-managed. The control components 206 generally provide  
25 management functions that perform tasks such as linking together appropriate core

1 layer components 208 for processing media. For example, the topology loader 224  
2 checks the multimedia data type of an incoming media file and determines which  
3 processing components (i.e., 230, 232, 234, 236, 238) of the core layer components  
4 208 need to be linked into a processing chain in order to properly render the data  
5 type. Note that for purposes of this disclosure, the media data type is DV data. A  
6 media engine component 226 of the control layer 206 manages the processing of  
7 the data through the chain of processing components (i.e., 230, 232, 234, 236, 238)  
8 assembled by the topology loader 224. For example, the media engine 226 pushes  
9 the data through the processing chain, controlling when to stop playback, start  
10 playback, play backwards, jump to a particular time, and so on.

11 Core layer components 208 include media sources 230, metadata read/write  
12 232, MUX/Dmux 234, transforms 236, and media sinks 238. Media sources 230  
13 provide multimedia data through a generic, well-defined interface. The media  
14 sources 230 describe the presentation, including video data streams to be accessed.  
15 There are many implementations of media sources for providing multimedia data  
16 from different multimedia file types or devices. However, the present disclosure is  
17 directed to multimedia in a DV format.

18 The transforms 236 of core layer 208 each perform some type of  
19 transformation operation on multimedia data through a generic, well-defined  
20 interface. Transform examples include codecs, DSPs, video resizers, audio  
21 resamplers, statistical processors, color resamplers, and others. Although the  
22 MUX/Dmux 234 (Dmux 234, hereinafter) is illustrated separately within core layer  
23 208, it is one representation of a transform 236 that takes interleaved multimedia  
24 data as an input, and separates the data into individually useful media streams of  
25 multimedia data. Thus, in the context of this disclosure, the Dmux 234 is a DV

1 Dmux 234 that, among other things, splits out the video and audio components of  
2 DV frames or samples from a DV media source 230.

3 The Dmux 234 is also illustrated within the multimedia architecture 202 of  
4 Fig. 2 as including DV metadata extraction tool 200. As described in further detail  
5 herein below, Dmux 234 supports DV metadata extraction through the DV  
6 metadata extraction tool 200. The DV metadata extraction tool 200 generally  
7 allows the user to create and manage an extraction list of DV Metadata packs to be  
8 extracted from a DV data stream. Once a DV metadata pack ID is added to the  
9 extraction list, the DV Dmux 234 extracts the associated DV metadata pack from  
10 each DV frame as it splits out the video and audio components of the DV frame.  
11 The DV Dmux 234 stores the DV metadata pack in a container and attaches the  
12 container to the outgoing video sample as an extended attribute. Although the DV  
13 metadata extraction tool 200 is discussed herein in conjunction with, or as a part  
14 of, Dmux 234, this is not intended as a limitation as to where the DV metadata  
15 extraction tool 200 can be implemented within the core layer 208 or elsewhere.  
16 Implementing the DV metadata extraction tool 200 within the Dmux 234 is a  
17 preferred embodiment because of the benefits of efficiency provided by the  
18 splitting function of the Dmux 234. Thus, the DV metadata extraction tool 200  
19 may just as easily be part of the media source 230, a DMO (DirectX Media  
20 Object), or a stand-alone software component anywhere else that has access to the  
21 DV data stream. The DV metadata extraction process is discussed in greater detail  
22 below with reference to subsequent figures.

23 Media sinks (sinks) 238 are also included in the core layer 208 processing  
24 components. Sinks 238 generally accept multimedia data as input through a  
25 generic, well-defined interface. There are many implementations of media sinks

1 for performing different functions with multimedia data, such as writing  
2 multimedia data to a given file type or to a network, or displaying the multimedia  
3 data on a video monitor using a video card.

4 The base components 210 and development platform 212 of multimedia  
5 architecture 202 generally make up mostly un-managed API's. The base  
6 components 210 include media container 240, networking 242, DRM 244, MPEG  
7 format support 246, and audio/video engine 248. These components generally  
8 perform individual functions in support of multimedia architecture 202. The  
9 development platform 212 generally includes resource management infrastructure  
10 and common primitive types such as samples, clocks, events, buffers, and so on.  
11 The definitions layer of multimedia architecture 202 includes definitions and  
12 policies related to schemas, protocols, and formats (e.g., metadata, device models,  
13 types, etc.).

14 Fig. 3 illustrates an example of DV data being processed through various  
15 processing components of the core layer 208, including the DV metadata  
16 extraction tool 200 of Dmux 234, as briefly discussed above. DV data samples  
17 (DV frames) 300 enter the Dmux 234 where they are split into video samples 302  
18 and audio samples 304. The video samples 302 proceed through the processing  
19 chain to various processing components such as video codec 308 and video  
20 renderer 310, after which they might be displayed on a video display 312. The  
21 audio samples 304 proceed through the processing chain to various processing  
22 components such as audio renderer 314, after which they might be played through  
23 an audio speaker 316. While the Dmux 234 is splitting out the DV samples 300, it  
24 also extracts DV metadata packs that it locates within the DV samples 300 in  
25 accordance with DV metadata pack IDs (DVPackIDs) from an extraction list (see

1 Fig. 4). Upon locating a DV metadata pack whose DVPackID is in the extraction  
2 list, the Dmux 234 extracts the DV metadata pack and stores it in a container 306  
3 and attaches the container 306 to the corresponding outgoing video sample 302 as  
4 an extended attribute.

5 For a certain subset of DV metadata packs, the DV metadata extraction tool  
6 200 also provides extended support of DV pack-specific data structures, called  
7 DV\_METADATA structures (see Fig. 5). In addition to storing the DV metadata  
8 packs in the container 306 for these extended support packs, the Dmux 234 also  
9 stores the unpacked DV\_METADATA structures in the container 306. Thus, for  
10 certain extended support DV metadata packs, the DV metadata extraction tool 200  
11 breaks down the packed data into usable DV pack-specific data structures, or  
12 DV\_METADATA structures. Fig. 5 illustrates a container 306 having one or  
13 more DV metadata packs 500 and one or more unpacked DV\_METADATA  
14 structures 502 that correspond with the DV metadata packs 500.

15 According to IEC 61834-4, there are 256 DV metadata packs in the DV  
16 format. The 256 DV metadata packs are shown herein below in a reference  
17 section of this disclosure entitled Interface Definition Language. Although 256  
18 packs are reserved for DV metadata, many of the packs have yet to be defined.  
19 The binary pack layout for various DV metadata packs is shown in the Interface  
20 Definition Language reference section. The DV metadata pack binary layouts  
21 included are for those DV metadata packs that are specifically supported as  
22 unpacked DV pack-specific data structures (i.e., DV\_METADATA structures).  
23 Thus, the Interface Definition Language section also includes the unpacked  
24 DV\_METADATA structures for the specifically supported DV metadata packs. In  
25 general, each DV metadata pack consists of 5 bytes in its binary layout. The first

1 byte in each DV metadata pack is the DVPackID, and the next four bytes consist  
2 of binary fields.

3 Referring again to Figs. 3 and 4, the DV metadata extraction tool 200  
4 supports an extraction API 400 (application programming interface) that maintains  
5 the extraction list 404 through various methods. The DV metadata extraction tool  
6 200 also supports a container API 402 that will be discussed below. Fig. 4  
7 illustrates the DV metadata extraction tool 200 along with the extraction API 400  
8 and container API 402 it supports. Also shown in Fig. 4 is the extraction list 404,  
9 which may contain various DVPackIDs. The extraction API 400 is called the  
10 IMFExtractDVMetadata API, and the methods it supports include AddPack,  
11 RemovePack, RemoveAllPacks, GetCount, and GetPack.

12 The AddPack method adds the specified pack to the extraction list 404 of  
13 DV Packs to be extracted on each DV frame 300 according to the following  
14 syntax:

15 **HRESULT AddPack(  
16 BYTE DVPackID  
17 );**

18 *DVPackID* is an input parameter that specifies the PackID for a DV  
19 metadata pack. This is a member of the DVPACKID enum. In a resulting  
20 DV\_METADATA structure the PackID is in DvMetadata.Pack[0]. The only pack  
21 that cannot be added to the extraction list 404 is DVPAC\_NO\_INFO (0xFF) (see  
22 the Interface Definition Language section).

23 If the AddPack method succeeds, it returns S\_OK. However, an  
24 E\_INVALIDARG will be returned if the DVPackID is DVPACK\_NO\_INFO.  
25 Other errors may also be returned.

1 A call to AddPack from an editing application 204, for example, adds a  
2 DVPackID to the extraction list 404. The function will succeed even if the pack  
3 (i.e., the DVPackID) has previously been added. The packs are not reference  
4 counted so a pack needs only to be removed once even if it has been added twice.

5 The RemovePack method removes the specified pack from the extraction  
6 list 404 of packs to be extracted on each DV frame 300 according to the following  
7 syntax:

8  
9 **HRESULT RemovePack(  
10 BYTE DVPackID  
11 );**

12 DVPackID is an input parameter that specifies the PackID for a DV  
13 metadata pack. This is a member of the DVPACKID enum. In a resulting  
14 DV\_METADATA structure, the PackID is in DvMetadata.Pack[0].

15 If the RemovePack method succeeds, it returns S\_OK. If the pack is not in  
16 the extraction list 404 then the function returns E\_ITEM\_NOT\_FOUND. Other  
17 error codes may also be returned.

18 A call to RemovePack from an editing application 204, for example,  
19 removes the specified pack from the extraction list 404.

20 The RemoveAllPacks method clears the extraction list 404 of DV Packs  
21 that the Dmux 234 would extract according to the following syntax:

22 **HRESULT RemoveAllPacks();**

23 There are no parameters input with the RemoveAllPack method. If the  
24 method succeeds, it returns S\_OK. Other error codes may also be returned.  
25 Calling RemoveAllPack, by an editing application 204, for example, clears the  
entire extraction list 404.

1       The GetCount method returns the number of DV packs that are in the  
2 extraction list 404 according to the following syntax:

3       **HRESULT GetCount(  
4        DWORD\* pCount  
5        );**

6       The *pCount* parameter is an output parameter that specifies the number of  
7 packs in the extraction list 404. If the method succeeds, it returns S\_OK. Calling  
8 GetCount retrieves the number of items (i.e., DVPackID's) in the extraction list  
9 404.

10       The GetPackID method returns the DVPackID of a pack at a given index in  
11 the extraction list 404 according to the following syntax:

12       **HRESULT GetPack(  
13        DWORD Index,  
14        BYTE\* pDVPackID  
15        );**

16       The Index parameter is an input parameter that is the index in the extraction  
17 list 404 of the desired DVPack ID. The *pDVPackID* is an output parameter that  
18 is a pointer to a byte where the object will copy the DVPack ID of the item found  
19 at the specified index.

20       If the GetPackID method succeeds, it returns S\_OK. If the Index is out of  
21 range, the method returns the error code, MF\_E\_INVALIDINDEX. If an error is  
22 returned the value OF pDVPackID is DVPACK\_NO\_INFO (0xFF).

23       The GetPackID method allows the caller (e.g., application 204) to retrieve  
24 the full list of items to be extracted by repeatedly calling GetPackId and  
25 incrementing the index until E\_INVALIDARG is returned.



1 As mentioned above, the DV metadata extraction tool 200 also supports a  
2 container API 402 (see Fig. 4). The container 306 (Fig. 3) is placed as a sample  
3 attribute on the video sample 302 that is split out by the Dmux 234. The container  
4 API 400 is called the IMFDVMetadataContainer API, and the methods it supports  
5 include Add, Remove, RemoveAll, GetCount, Lock, Unlock, GetFirst and  
6 GetNext. In general, the IMFDVMetadataContainer API provides a general  
7 mechanism for adding attributes to the list, removing attributes from the list,  
8 clearing the container 306 and iterating through the container 306.

9 The Add method adds a DV pack-specific data structure, or  
10 DV\_METADATA structure, to the container 306 according to the following  
11 syntax:

```
12 HRESULT Add(  
13 const DV_METADATA* pMetadata,  
14 UINT32* puIndex  
15 );
```

16 The *pMetadata* parameter is an input parameter that is a constant pointer to  
17 a DV\_METADATA structure. *pMetadata->cbSize* is used to allocate memory in  
18 the container 306 and a copy of the entire DV\_METADATA structure placed in  
19 the newly allocated memory.

20 The *ulIndex* is an output parameter that returns the index of the newly  
21 added DV\_METADATA structure. The index may change if additional structures  
22 are added or deleted from the container 306.

23 If the Add method succeeds, it returns S\_OK. It may also return  
24 E\_OUTOFMEMORY if it is unable to allocate sufficient space for the new item.  
25 This operation will complete in constant time O(k). This operation will block until

1 the lock is released if the container has been locked by another thread. (see Lock  
2 and Unlock methods below).

3 The Remove method removes a DV pack-specific data structure, or  
4 DV\_METADATA structure, from the container 306 according to the following  
5 syntax:

```
6     HRESULT Remove(  
7         UINT32 uIndex  
8     );
```

9 The *uIndex* parameter is an input parameter that indicates the index of the  
10 item that is to be removed from the container 306. When an item is removed from  
11 the container 306 the index of items that remains in the container 306 may change.

12 If the method succeeds, it returns S\_OK. It may also return  
13 E\_INVALIDARG if an item with a matching index cannot be found. This  
14 includes the case when the container 306 is empty. This operation will complete in  
15 linear time O(n), where n is the number of items stored in the list. This operation  
16 will block until the lock is released if the container has been locked by another  
17 thread. (see Lock and Unlock methods below).

18 The RemoveAll method clears all items (e.g., DV metadata packs and DV  
19 pack-specific data structures) from the container 306 according to the following  
20 syntax:

```
21     HRESULT RemoveAll( );
```

22 There are no parameters input with the RemoveAll method. If the method  
23 succeeds, it returns S\_OK and there will be no more items in the container 306.  
24 However, it does not necessarily follow that the memory will be freed. The  
25 container 306 may implement a pooling scheme to avoid repeated small

1 allocations. This operation will complete in linear time  $O(n)$ , where  $n$  is the  
2 number of items in the container 306. This operation will block until the lock is  
3 released if the lock has been acquired on another thread. (see Lock and Unlock  
4 methods below).

5 The GetCount method returns the count of items in the container 306  
6 according to the following syntax:

7 **HRESULT GetCount(  
8 **UINT32\* puCount  
9 );****

10 The *puCount* parameter is an output parameter that returns number of items  
11 currently in the container 306. If the method succeeds, it returns S\_OK.

12 This operation will complete in constant time  $O(k)$ . The count returned is  
13 only valid at the time that the call was made. Objects may be added or removed by  
14 other threads. Locking the object will prevent other threads from adding or  
15 removing items from the container until the lock is released. (see Lock and  
16 Unlock methods below).

17 The Lock method is used to lock the container 306 for exclusive access.  
18 This guarantees that the container 306 can be iterated and the returned pointers to  
19 DV\_METADATA structures will remain valid as long as the lock owner does not  
20 add or remove items. The syntax for this method is as follows:

21 **HRESULT Lock();**

22 There are no input parameters with the Lock method. If the method  
23 succeeds, it returns S\_OK. It may return other error codes. If the Lock is  
24 unavailable, the call will block until the lock can be acquired.  
25

1       The Unlock method releases the lock obtained via the Lock method  
2 according to the following syntax.

```
3       HRESULT Unlock()  
4       UINT32* puIndex,  
5       const DV_METADATA** pMetadata  
6       );
```

7       There are no input parameters with the Unlock method. If the method  
8 succeeds, it returns S\_OK. It may return other error codes.

9       The GetFirst method starts iterating from the beginning of the container 306  
10 according to the following syntax:

```
11       HRESULT GetFirst(  
12       UINT32* puIndex,  
13       Const DV_METADATA** ppMetadata  
14       );
```

15       The *puIndex* parameter is an output parameter that specifies the index of  
16 the item retrieved from the container 306. The *ppMetadata* parameter is an  
17 output parameter that specifies a pointer to the objects internal data structure  
18 containing the metadata. This pointer may be invalidated if items are added or  
19 removed from the container 306.

20       If the method succeeds, it returns S\_OK. The method may return  
21 E\_INVALIDARG if the index is out of range or the container 306 has had an item  
22 added or removed from it since the last call to GetFirst(). The method will return  
23 MF\_E\_INVALIDREQUEST if the object has not been locked. Calling the Lock  
24 method ensures that items are not added or removed from the container 306 by  
25 other threads while iterating the list.

26       The GetNext method iterates through each item in the container 306  
27 according to the syntax:

1       **HRESULT** GetNext(  
2       **UINT32\*** puIndex,  
3       **Const DV\_METADATA\*\*** ppMetadata  
4       );

5       The *puIndex* parameter is an output parameter that specifies the index of  
6       the item retrieved from the container 306. The *ppMetadata* parameter is an output  
7       parameter that specifies a pointer to the objects internal data structure containing  
8       the metadata. This pointer may be invalidated if items are added or removed from  
9       the container 306.

10       If the method succeeds, it returns **S\_OK**. The method may return  
11       **E\_INVALIDARG** if the index is out of range or the container 306 has had an item  
12       added or removed from it since the last call to **GetFirst()**. The method will return  
13       **MF\_E\_INVALIDREQUEST** if the object has not been locked. Calling the **Lock**  
14       method ensures that items are not added or removed from the container 306 by  
15       other threads while iterating the list.

### 16       **Exemplary Methods**

17       Example methods for extracting DV metadata from a DV data stream will  
18       now be described with primary reference to the flow diagrams of Figs. 6 - 7. The  
19       methods apply generally to the exemplary embodiments discussed above with  
20       respect to Figs. 2 - 5. The elements of the described methods may be performed by  
21       any appropriate means including, for example, by hardware logic blocks on an  
22       ASIC or by the execution of processor-readable instructions defined on a  
23       processor-readable medium.

24       A "processor-readable medium," as used herein, can be any means that can  
25       contain, store, communicate, propagate, or transport instructions for use by or

1 execution by a processor. A processor-readable medium can be, without  
2 limitation, an electronic, magnetic, optical, electromagnetic, infrared, or  
3 semiconductor system, apparatus, device, or propagation medium. More specific  
4 examples of a processor-readable medium include, among others, an electrical  
5 connection (electronic) having one or more wires, a portable computer diskette  
6 (magnetic), a random access memory (RAM) (magnetic), a read-only memory  
7 (ROM) (magnetic), an erasable programmable-read-only memory (EPROM or  
8 Flash memory), an optical fiber (optical), a rewritable compact disc (CD-RW)  
9 (optical), and a portable compact disc read-only memory (CDROM) (optical).

10 Fig. 6 shows an exemplary method 600 for extracting DV metadata from a  
11 DV data stream. At block 602, an instruction is received that specifies additional  
12 per-frame DV metadata to extract from a DV data stream. The instruction is  
13 received by a DV metadata extraction tool 200 that can be part of a multimedia  
14 architecture 202 on a computer 102. The instruction is typically received from an  
15 application 204, such as a DV editing application executing on computer 102. The  
16 instruction is directed to an extraction interface 400 of the DV metadata extraction  
17 tool 200 in the form of a method call supported by the extraction interface 400.  
18 The instruction identifies the DV metadata by a DV-PackID included within the  
19 method call. Method calls supported by the extraction interface 400 include an  
20 AddPack method call to add a DV-PackID to a DV metadata extraction list 404, a  
21 RemovePack method call to remove a DV-PackID from the extraction list 404, and  
22 a RemoveAllPacks method call to remove all DV-PackIDs from the extraction list  
23 404. Additional method calls supported by the extraction interface 400 include a  
24 GetCount method call that returns a number indicating an amount of DV-PackIDs  
25

1 present in the extraction list 404 and a GetPackID method call that returns a  
2 DVPackID at a specified index in the extraction list 404.

3 At block 604, the DV metadata specified in the instruction is extracted from  
4 a DV frame of the DV data stream. In one implementation, a Dmux 234 within a  
5 core layer 208 of the multimedia architecture 202 extracts the specified DV  
6 metadata as it splits the DV frame 300 into component video 302 and audio 304  
7 samples. The extraction includes the Dmux 234 looking at the DV metadata  
8 extraction list 404 to determine which DV metadata packs to extract. At block  
9 606, the DV metadata is stored in a container 306. At block 608, the container is  
10 attached to a video sample 302 split off of the DV frame 300 by the Dmux 234.

11 At block 610, the container is managed by the DV metadata extraction tool  
12 200. The DV metadata extraction tool 200 includes a container interface 402 that  
13 supports methods by which applications 204 can access and manage data in the  
14 container 306. Method calls supported by the container interface 402 are an Add  
15 method call that adds a DV\_METADATA structure to the container, a Remove  
16 method call that removes a DV\_METADATA structure from the container, a  
17 RemoveAll method call that removes all items from the container, a GetCount  
18 method call that returns a number indicating an amount of items present in the  
19 container, a Lock method call that locks the container for exclusive access, an  
20 Unlock method call that unlocks the container, a GetFirst method call that retrieves  
21 an item from the container at a beginning index of the container, and a GetNext  
22 method call that retrieves an item from the container at a next index of the  
23 container.

24 Fig. 7 shows another exemplary method 700 for extracting DV metadata  
25 from a DV data stream. At block 702, a DV metadata extraction list 404 is

1 managed. The extraction list 404 is managed by a DV metadata extraction tool  
2 200. The DV metadata extraction tool 200 supports an extraction interface 400 for  
3 managing the extraction list through various methods. Methods supported by the  
4 extraction interface 400 include an AddPack method call to add a DVPackID to a  
5 DV metadata extraction list 404, a RemovePack method call to remove a  
6 DVPackID from the extraction list 404, and a RemoveAllPacks method call to  
7 remove all DVPackIDs from the extraction list 404. Additional methods supported  
8 by the extraction interface 400 include a GetCount method call that returns a  
9 number indicating an amount of DVPackIDs present in the extraction list 404 and  
10 a GetPackID method call that returns a DVPackID at a specified index in the  
11 extraction list 404.

12 At block 704, a DV metadata pack is extracted from a DV frame 300 based  
13 on a DVPackID in the extraction list 404. In one implementation, a Dmux 234  
14 within a core layer 208 of the multimedia architecture 202 extracts the specified  
15 DV metadata pack as it splits the DV frame 300 into component video 302 and  
16 audio 304 samples. The extraction includes the Dmux 234 looking at the DV  
17 metadata extraction list 404 to determine which DV metadata packs to extract. At  
18 block 706, the DV metadata pack is stored in an IMFDVMetadataContainer 306.

19 At block 708, the DV metadata pack is unpacked into a DV pack-specific  
20 data structure. The DV pack-specific data structure breaks out the packed binary  
21 data from the DV metadata pack and assigns binary values to corresponding  
22 variable names, making it easy for an application program 204 to utilize the data  
23 from the DV metadata pack. At block 710, the DV pack-specific data structure is  
24 stored in the IMFDVMetadataContainer 306, and at block 712, the  
25



1 IMFDMetadataContainer 306 is attached to a DV video sample 302 split off of  
2 the DV frame 300 by the Dmux 234.

3 At block 714, the IMFDMetadataContainer 306 is managed by the DV  
4 metadata extraction tool 200. The DV metadata extraction tool 200 includes a  
5 container interface 402 that supports methods by which applications 204 can  
6 access and manage data in the container 306. Method calls supported by the  
7 container interface 402 are an Add method call that adds a DV pack-specific data  
8 structure (called a DV\_METADATA structure) to the container, a Remove method  
9 call that removes a DV\_METADATA structure from the container, a RemoveAll  
10 method call that removes all items from the container, a GetCount method call that  
11 returns a number indicating an amount of items present in the container, a Lock  
12 method call that locks the container for exclusive access, an Unlock method call  
13 that unlocks the container, a GetFirst method call that retrieves an item from the  
14 container at a beginning index of the container, and a GetNext method call that  
15 retrieves an item from the container at a next index of the container.

16 While one or more methods have been disclosed by means of flow diagrams  
17 and text associated with the blocks of the flow diagrams, it is to be understood that  
18 the blocks do not necessarily have to be performed in the order in which they were  
19 presented, and that an alternative order may result in similar advantages.  
20 Furthermore, the methods are not exclusive and can be performed alone or in  
21 combination with one another.

## Exemplary Computer

Fig. 8 illustrates an exemplary computing environment 800 suitable for implementing a computer 102. Although one specific configuration is shown, client computer 102 may be implemented in other computing configurations.

The computing environment 800 includes a general-purpose computing system in the form of a computer 802. The components of computer 802 can include, but are not limited to, one or more processors or processing units 804, a system memory 806, and a system bus 808 that couples various system components including the processor 804 to the system memory 806.

The system bus 808 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. An example of a system bus 808 would be a Peripheral Component Interconnects (PCI) bus, also known as a Mezzanine bus.

Computer 802 typically includes a variety of computer readable media. Such media can be any available media that is accessible by computer 802 and includes both volatile and non-volatile media, removable and non-removable media. The system memory 806 includes computer readable media in the form of volatile memory, such as random access memory (RAM) 810, and/or non-volatile memory, such as read only memory (ROM) 812. A basic input/output system (BIOS) 814, containing the basic routines that help to transfer information between elements within computer 802, such as during start-up, is stored in ROM 812. RAM 810 typically contains data and/or program modules that are immediately accessible to and/or presently operated on by the processing unit 804.

1 Computer 802 can also include other removable/non-removable,  
2 volatile/non-volatile computer storage media. By way of example, Fig. 8  
3 illustrates a hard disk drive 816 for reading from and writing to a non-removable,  
4 non-volatile magnetic media (not shown), a magnetic disk drive 818 for reading  
5 from and writing to a removable, non-volatile magnetic disk 820 (e.g., a "floppy  
6 disk"), and an optical disk drive 822 for reading from and/or writing to a  
7 removable, non-volatile optical disk 824 such as a CD-ROM, DVD-ROM, or other  
8 optical media. The hard disk drive 816, magnetic disk drive 818, and optical disk  
9 drive 822 are each connected to the system bus 808 by one or more data media  
10 interfaces 826. Alternatively, the hard disk drive 816, magnetic disk drive 818, and  
11 optical disk drive 822 can be connected to the system bus 808 by a SCSI interface  
12 (not shown).

13 The disk drives and their associated computer-readable media provide non-  
14 volatile storage of computer readable instructions, data structures, program  
15 modules, and other data for computer 802. Although the example illustrates a hard  
16 disk 816, a removable magnetic disk 820, and a removable optical disk 824, it is to  
17 be appreciated that other types of computer readable media which can store data  
18 that is accessible by a computer, such as magnetic cassettes or other magnetic  
19 storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or  
20 other optical storage, random access memories (RAM), read only memories  
21 (ROM), electrically erasable programmable read-only memory (EEPROM), and  
22 the like, can also be utilized to implement the exemplary computing system and  
23 environment.

24 Any number of program modules can be stored on the hard disk 816,  
25 magnetic disk 820, optical disk 824, ROM 812, and/or RAM 810, including by

1 way of example, an operating system 826, one or more application programs 828,  
2 other program modules 830, and program data 832. Each of such operating system  
3 826, one or more application programs 828, other program modules 830, and  
4 program data 832 (or some combination thereof) may include an embodiment of a  
5 caching scheme for user network access information.

6 Computer 802 can include a variety of computer/processor readable media  
7 identified as communication media. Communication media typically embodies  
8 computer readable instructions, data structures, program modules, or other data in  
9 a modulated data signal such as a carrier wave or other transport mechanism and  
10 includes any information delivery media. The term “modulated data signal” means  
11 a signal that has one or more of its characteristics set or changed in such a manner  
12 as to encode information in the signal. By way of example, and not limitation,  
13 communication media includes wired media such as a wired network or direct-  
14 wired connection, and wireless media such as acoustic, RF, infrared, and other  
15 wireless media. Combinations of any of the above are also included within the  
16 scope of computer readable media.

17 A user can enter commands and information into computer system 802 via  
18 input devices such as a keyboard 834 and a pointing device 836 (e.g., a “mouse”).  
19 Other input devices 838 (not shown specifically) may include a microphone,  
20 joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and  
21 other input devices are connected to the processing unit 804 via input/output  
22 interfaces 840 that are coupled to the system bus 808, but may be connected by  
23 other interface and bus structures, such as a parallel port, game port, or a universal  
24 serial bus (USB).

1 A monitor 842 or other type of display device can also be connected to the  
2 system bus 808 via an interface, such as a video adapter 844. In addition to the  
3 monitor 842, other output peripheral devices can include components such as  
4 speakers (not shown) and a printer 846 which can be connected to computer 802  
5 via the input/output interfaces 840.

6 Computer 802 can operate in a networked environment using logical  
7 connections to one or more remote computers, such as a remote computing device  
8 848. By way of example, the remote computing device 848 can be a personal  
9 computer, portable computer, a server, a router, a network computer, a peer device  
10 or other common network node, and the like. The remote computing device 848 is  
11 illustrated as a portable computer that can include many or all of the elements and  
12 features described herein relative to computer system 802.

13 Logical connections between computer 802 and the remote computer 848  
14 are depicted as a local area network (LAN) 850 and a general wide area network  
15 (WAN) 852. Such networking environments are commonplace in offices,  
16 enterprise-wide computer networks, intranets, and the Internet. When  
17 implemented in a LAN networking environment, the computer 802 is connected to  
18 a local network 850 via a network interface or adapter 854. When implemented in  
19 a WAN networking environment, the computer 802 typically includes a modem  
20 856 or other means for establishing communications over the wide network 852.  
21 The modem 856, which can be internal or external to computer 802, can be  
22 connected to the system bus 808 via the input/output interfaces 840 or other  
23 appropriate mechanisms. It is to be appreciated that the illustrated network  
24 connections are exemplary and that other means of establishing communication  
25 link(s) between the computers 802 and 848 can be employed.

1 In a networked environment, such as that illustrated with computing  
2 environment 800, program modules depicted relative to the computer 802, or  
3 portions thereof, may be stored in a remote memory storage device. By way of  
4 example, remote application programs 858 reside on a memory device of remote  
5 computer 848. For purposes of illustration, application programs and other  
6 executable program components, such as the operating system, are illustrated  
7 herein as discrete blocks, although it is recognized that such programs and  
8 components reside at various times in different storage components of the  
9 computer system 802, and are executed by the data processor(s) of the computer.

#### 10 **Interface Definition Language**

11 As indicated above, this IDL (Interface Definition Language) section lists  
12 the 256 DV metadata packs as well as the binary pack layout for various of those  
13 packs specifically supported in an extended manner as DV pack-specific data  
14 structures (i.e., DV\_METADATA structures) by the DV metadata extraction tool  
15 200. This section also includes the specific layouts of the unpacked DV metadata  
16 packs for the supported DV\_METADATA structures. The DV metadata packs  
17 that have extended support are identified in the following table:

18 **TABLE 1**

19	CONTROL	
20	CASSETTE ID	0X00
21	TAPE LENGTH	0x01
22	TEXT HEADER	0x08
23	TEXT	0x09
24	TAG	0x0B
25		
	TITLE	
	TIME CODE	0x13
	BINARY GROUP	0x14
	TEXT HEADER	0x18

	TEXT	0x19
PROGRAM		
	PROGRAM REC DTIME	0x42
AAUX		
	SOURCE	0x50
	SOURCE CONTROL	0x51
	REC DATE	0x52
	REC TIME	0x53
	BINARY GROUP	0x54
	CLOSED CAPTION	0x55
	TR	0x56
VAUX		
	SOURCE	0x60
	SOURCE CONTROL	0x61
	REC DATE	0x62
	REC TIME	0x63
	BINARY GROUP	0x64
	CLOSED CAPTION	0x65
	TR	0x66
CAMERA		
	CONSUMER CAMERA 1	0x70
	CONSUMER CAMERA 2	0x71
	CAMERA SHUTTER	0x7F

Each DV pack-specific data structure (i.e., DV\_METADATA structure) that has extended support by the DV metadata extraction tool 200 starts with a size and a DV Pack. The size member contains the size of the complete DV\_METADATA structure. The DVPack (5 byte array) is the raw DV metadata. Each pack consists of 5 bytes. The first byte is the pack ID from Table 1 above. The next four bytes contain bit-fields containing the data. Each of the extended

support packs has an associated structure where the bit-fields are unpacked and lightly processed into a more useable form. The full definition of the DV Packs is found in IEC 61834-4.

The 256 DV metadata packs and DV pack-specific data structures (i.e., DV\_METADATA structures) supported by the DV metadata extraction tool 200 are as follows:

```
typedef enum _DVPACKID
{
    DVPACK_CONTROL_CASSETTE_ID =          0x00,
    DVPACK_CONTROL_TAPE_LENGTH =          0x01,
    DVPACK_CONTROL_TIMER_ACT_DATE =        0x02,
    DVPACK_CONTROL_TIMER_ACS_S_S =         0x03,
    DVPACK_CONTROL_PR_START_POINT_04 =     0x04,
    DVPACK_CONTROL_PR_START_POINT_05 =     0x05,
    DVPACK_CONTROL_TAG_ID_NO_GENRE =       0x06,
    DVPACK_CONTROL_TOPIC_PAGE_HEADER =     0x07,
    DVPACK_CONTROL_TEXT_HEADER =           0x08,
    DVPACK_CONTROL_TEXT =                  0x09,
    DVPACK_CONTROL_TAG_0A =                 0x0A,
    DVPACK_CONTROL_TAG_0B =                 0x0B,
    DVPACK_CONTROL_TELETEXT_INFO =         0x0C,
    DVPACK_CONTROL_KEY =                   0x0D,
    DVPACK_CONTROL_ZONE_END_0E =           0x0E,
    DVPACK_CONTROL_ZONE_END_0F =           0x0F,
    DVPACK_TITLE_TOTAL_TIME =              0x10,
    DVPACK_TITLE_REMAIN_TIME =             0x11,
    DVPACK_TITLE_CHAPTER_TOTAL_NO =        0x12,
    DVPACK_TITLE_TIME_CODE =               0x13,
    DVPACK_TITLE_BINARY_GROUP =            0x14,
    DVPACK_TITLE_CASSETTE_NO =             0x15,
    DVPACK_TITLE_SOFT_ID_16 =              0x16,
    DVPACK_TITLE_SOFT_ID_17 =              0x17,
    DVPACK_TITLE_TEXT_HEADER =             0x18,
    DVPACK_TITLE_TEXT =                    0x19,
    DVPACK_TITLE_TITLE_START_1A =          0x1A,
    DVPACK_TITLE_TITLE_START_1B =          0x1B,
    DVPACK_TITLE_REEL_ID_1C =              0x1C,
    DVPACK_TITLE_REEL_ID_1D =              0x1D,
    DVPACK_TITLE_TITLE_END_1E =            0x1E,
    DVPACK_TITLE_TITLE_END_1F =            0x1F,
    DVPACK_CHAPTER_TOTAL_TIME =            0x20,
    DVPACK_CHAPTER_REMAIN_TIME =           0x21,
```



1	DVPACK_CHAPTER_CHAPTER_NO =	0x22,
	DVPACK_CHAPTER_TIME_CODE =	0x23,
	DVPACK_CHAPTER_BINARY_GROUP =	0x24,
2	DVPACK_CHAPTER_RESERVED_25 =	0x25,
	DVPACK_CHAPTER_RESERVED_26 =	0x26,
3	DVPACK_CHAPTER_RESERVED_27 =	0x27,
	DVPACK_CHAPTER_TEXT_HEADER =	0x28,
4	DVPACK_CHAPTER_TEXT =	0x29,
	DVPACK_CHAPTER_CHAPTER_START_2A =	0x2A,
5	DVPACK_CHAPTER_CHAPTER_START_2B =	0x2B,
	DVPACK_CHAPTER_RESERVED_2C =	0x2C,
6	DVPACK_CHAPTER_RESERVED_2D =	0x2D,
	DVPACK_CHAPTER_CHAPTER_END_2E =	0x2E,
7	DVPACK_CHAPTER_CHAPTER_END_2F =	0x2F,
	DVPACK_PART_TOTAL_TIME =	0x30,
8	DVPACK_PART_REMAIN_TIME =	0x31,
	DVPACK_PART_PART_NO =	0x32,
9	DVPACK_PART_TIME_CODE =	0x33,
	DVPACK_PART_BINARY_GROUP =	0x34,
10	DVPACK_PART_RESERVED_35 =	0x35,
	DVPACK_PART_RESERVED_36 =	0x36,
11	DVPACK_PART_RESERVED_37 =	0x37,
	DVPACK_PART_TEXT_HEADER =	0x38,
12	DVPACK_PART_TEXT =	0x39,
	DVPACK_PART_START_3A =	0x3A,
13	DVPACK_PART_START_3B =	0x3B,
	DVPACK_PART_RESERVED_3C =	0x3C,
14	DVPACK_PART_RESERVED_3D =	0x3D,
	DVPACK_PART_PART_END_3E =	0x3E,
15	DVPACK_PART_PART_END_3F =	0x3F,
	DVPACK_PROGRAM_TOTAL_TIME =	0x40,
16	DVPACK_PROGRAM_REMAIN_TIME =	0x41,
	DVPACK_PROGRAM_REC_DTIME =	0x42,
17	DVPACK_PROGRAM_TIME_CODE =	0x43,
	DVPACK_PROGRAM_BINARY_GROUP =	0x44,
18	DVPACK_PROGRAM_RESERVED_45 =	0x45,
	DVPACK_PROGRAM_RESERVED_46 =	0x46,
19	DVPACK_PROGRAM_RESERVED_47 =	0x47,
	DVPACK_PROGRAM_TEXT_HEADER =	0x48,
20	DVPACK_PROGRAM_TEXT =	0x49,
	DVPACK_PROGRAM_PROGRAM_START_4A =	0x4A,
21	DVPACK_PROGRAM_PROGRAM_START_4B =	0x4B,
	DVPACK_PROGRAM_RESERVED_4C =	0x4C,
22	DVPACK_PROGRAM_RESERVED_4D =	0x4D,
	DVPACK_PROGRAM_PROGRAM_END_4E =	0x4E,
23	DVPACK_PROGRAM_PROGRAM_END_4F =	0x4F,
	DVPACK_AAUX_SOURCE =	0x50,
24	DVPACK_AAUX_SOURCE_CONTROL =	0x51,
	DVPACK_AAUX_REC_DATE =	0x52,
25	DVPACK_AAUX_REC_TIME =	0x53,

1	DVPACK_AAUX_BINARY_GROUP =	0x54,
	DVPACK_AAUX_CLOSED_CAPTION =	0x55,
	DVPACK_AAUX_TR =	0x56,
2	DVPACK_AAUX_RESERVED_57 =	0x57,
	DVPACK_AAUX_TEXT_HEADER =	0x58,
3	DVPACK_AAUX_TEXT =	0x59,
	DVPACK_AAUX_AAUX_START_5A =	0x5A,
4	DVPACK_AAUX_AAUX_START_5B =	0x5B,
	DVPACK_AAUX_RESERVED_5C =	0x5C,
5	DVPACK_AAUX_RESERVED_5D =	0x5D,
	DVPACK_AAUX_AAUX_END_5E =	0x5E,
6	DVPACK_AAUX_AAUX_END_5F =	0x5F,
	DVPACK_VAUX_SOURCE =	0x60,
7	DVPACK_VAUX_SOURCE_CONTROL =	0x61,
	DVPACK_VAUX_REC_DATE =	0x62,
8	DVPACK_VAUX_REC_TIME =	0x63,
	DVPACK_VAUX_BINARY_GROUP =	0x64,
9	DVPACK_VAUX_CLOSED_CAPTION =	0x65,
	DVPACK_VAUX_TR =	0x66,
10	DVPACK_VAUX_TELETEXT =	0x67,
	DVPACK_VAUX_TEXT_HEADER =	0x68,
11	DVPACK_VAUX_TEXT =	0x69,
	DVPACK_VAUX_VAUX_START_6A =	0x6A,
12	DVPACK_VAUX_VAUX_START_6B =	0x6B,
	DVPACK_VAUX_MARINE_MOUNTAIN =	0x6C,
13	DVPACK_VAUX_LONGITUDE_LATITUDE =	0x6D,
	DVPACK_VAUX_VAUX_END_6E =	0x6E,
14	DVPACK_VAUX_VAUX_END_6F =	0x6F,
	DVPACK_CAMERA_CONSUMER_CAMERA_1 =	0x70,
15	DVPACK_CAMERA_CONSUMER_CAMERA_2 =	0x71,
	DVPACK_CAMERA_RESERVED_72 =	0x72,
16	DVPACK_CAMERA_LENS =	0x73,
	DVPACK_CAMERA_GAIN =	0x74,
17	DVPACK_CAMERA_PEDESTAL =	0x75,
	DVPACK_CAMERA_GAMMA =	0x76,
18	DVPACK_CAMERA_DETAIL =	0x77,
	DVPACK_CAMERA_TEXT_HEADER =	0x78,
19	DVPACK_CAMERA_TEXT =	0x79,
	DVPACK_CAMERA_RESERVED_7A =	0x7A,
20	DVPACK_CAMERA_CAMERA_PRESET =	0x7B,
	DVPACK_CAMERA_FLARE =	0x7C,
21	DVPACK_CAMERA_SHADING =	0x7D,
	DVPACK_CAMERA_KNEE =	0x7E,
22	DVPACK_CAMERA_SHUTTER =	0x7F,
	DVPACK_LINE_HEADER =	0x80,
23	DVPACK_LINE_Y =	0x81,
	DVPACK_LINE_CR =	0x82,
24	DVPACK_LINE_CB =	0x83,
	DVPACK_LINE_RESERVED_84 =	0x84,
25	DVPACK_LINE_RESERVED_85 =	0x85,

1	DVPACK_LINE_RESERVED_86 =	0x86,
	DVPACK_LINE_RESERVED_87 =	0x87,
	DVPACK_LINE_TEXT_HEADER =	0x88,
2	DVPACK_LINE_TEXT =	0x89,
	DVPACK_LINE_LINE_START_8A =	0x8A,
3	DVPACK_LINE_LINE_START_8B =	0x8B,
	DVPACK_LINE_RESERVED_8C =	0x8C,
4	DVPACK_LINE_RESERVED_8D =	0x8D,
	DVPACK_LINE_LINE_END_8E =	0x8E,
5	DVPACK_LINE_LINE_END_8F =	0x8F,
	DVPACK_MPEG_SOURCE =	0x90,
6	DVPACK_MPEG_SOURCE_CONTROL =	0x91,
	DVPACK_MPEG_REC_DATE =	0x92,
7	DVPACK_MPEG_REC_TIME =	0x93,
	DVPACK_MPEG_BINARY_GROUP =	0x94,
8	DVPACK_MPEG_STREAM =	0x95,
	DVPACK_MPEG_RESERVED_96 =	0x96,
9	DVPACK_MPEG_RESERVED_97 =	0x97,
	DVPACK_MPEG_TEXT_HEADER =	0x98,
10	DVPACK_MPEG_TEXT =	0x99,
	DVPACK_MPEG_SERVICE_START_9A =	0x9A,
11	DVPACK_MPEG_SERVICE_START_9B =	0x9B,
	DVPACK_MPEG_RESERVED_9C =	0x9C,
12	DVPACK_MPEG_RESERVED_9D =	0x9D,
	DVPACK_MPEG_SERVICE_END_9E =	0x9E,
13	DVPACK_MPEG_SERVICE_END_9F =	0x9F,
	DVPACK_RESERVED_RESERVED_A0 =	0xA0,
14	DVPACK_RESERVED_RESERVED_A1 =	0xA1,
	DVPACK_RESERVED_RESERVED_A2 =	0xA2,
15	DVPACK_RESERVED_RESERVED_A3 =	0xA3,
	DVPACK_RESERVED_RESERVED_A4 =	0xA4,
16	DVPACK_RESERVED_RESERVED_A5 =	0xA5,
	DVPACK_RESERVED_RESERVED_A6 =	0xA6,
17	DVPACK_RESERVED_RESERVED_A7 =	0xA7,
	DVPACK_RESERVED_RESERVED_A8 =	0xA8,
18	DVPACK_RESERVED_RESERVED_A9 =	0xA9,
	DVPACK_RESERVED_RESERVED_AA =	0xAA,
19	DVPACK_RESERVED_RESERVED_AB =	0xAB,
	DVPACK_RESERVED_RESERVED_AC =	0xAC,
20	DVPACK_RESERVED_RESERVED_AD =	0xAD,
	DVPACK_RESERVED_RESERVED_AE =	0xAE,
21	DVPACK_RESERVED_RESERVED_AF =	0xAF,
	DVPACK_RESERVED_RESERVED_B0 =	0xB0,
22	DVPACK_RESERVED_RESERVED_B1 =	0xB1,
	DVPACK_RESERVED_RESERVED_B2 =	0xB2,
23	DVPACK_RESERVED_RESERVED_B3 =	0xB3,
	DVPACK_RESERVED_RESERVED_B4 =	0xB4,
24	DVPACK_RESERVED_RESERVED_B5 =	0xB5,
	DVPACK_RESERVED_RESERVED_B6 =	0xB6,
25	DVPACK_RESERVED_RESERVED_B7 =	0xB7,

1	DVPACK_RESERVED_RESERVED_B8 =	0xB8,
	DVPACK_RESERVED_RESERVED_B9 =	0xB9,
	DVPACK_RESERVED_RESERVED_BA =	0xBA,
2	DVPACK_RESERVED_RESERVED_BB =	0xBB,
	DVPACK_RESERVED_RESERVED_BC =	0xBC,
3	DVPACK_RESERVED_RESERVED_BD =	0xBD,
	DVPACK_RESERVED_RESERVED_BE =	0xBE,
4	DVPACK_RESERVED_RESERVED_BF =	0xBF,
	DVPACK_RESERVED_RESERVED_C0 =	0xC0,
5	DVPACK_RESERVED_RESERVED_C1 =	0xC1,
	DVPACK_RESERVED_RESERVED_C2 =	0xC2,
6	DVPACK_RESERVED_RESERVED_C3 =	0xC3,
	DVPACK_RESERVED_RESERVED_C4 =	0xC4,
7	DVPACK_RESERVED_RESERVED_C5 =	0xC5,
	DVPACK_RESERVED_RESERVED_C6 =	0xC6,
8	DVPACK_RESERVED_RESERVED_C7 =	0xC7,
	DVPACK_RESERVED_RESERVED_C8 =	0xC8,
9	DVPACK_RESERVED_RESERVED_C9 =	0xC9,
	DVPACK_RESERVED_RESERVED_CA =	0xCA,
10	DVPACK_RESERVED_RESERVED_CB =	0xCB,
	DVPACK_RESERVED_RESERVED_CC =	0xCC,
11	DVPACK_RESERVED_RESERVED_CD =	0xCD,
	DVPACK_RESERVED_RESERVED_CE =	0xCE,
12	DVPACK_RESERVED_RESERVED_CF =	0xCF,
	DVPACK_RESERVED_RESERVED_D0 =	0xD0,
13	DVPACK_RESERVED_RESERVED_D1 =	0xD1,
	DVPACK_RESERVED_RESERVED_D2 =	0xD2,
14	DVPACK_RESERVED_RESERVED_D3 =	0xD3,
	DVPACK_RESERVED_RESERVED_D4 =	0xD4,
15	DVPACK_RESERVED_RESERVED_D5 =	0xD5,
	DVPACK_RESERVED_RESERVED_D6 =	0xD6,
16	DVPACK_RESERVED_RESERVED_D7 =	0xD7,
	DVPACK_RESERVED_RESERVED_D8 =	0xD8,
17	DVPACK_RESERVED_RESERVED_D9 =	0xD9,
	DVPACK_RESERVED_RESERVED_DA =	0xDA,
18	DVPACK_RESERVED_RESERVED_DB =	0xDB,
	DVPACK_RESERVED_RESERVED_DC =	0xDC,
19	DVPACK_RESERVED_RESERVED_DD =	0xDD,
	DVPACK_RESERVED_RESERVED_DE =	0xDE,
20	DVPACK_RESERVED_RESERVED_DF =	0xDF,
	DVPACK_RESERVED_RESERVED_E0 =	0xE0,
21	DVPACK_RESERVED_RESERVED_E1 =	0xE1,
	DVPACK_RESERVED_RESERVED_E2 =	0xE2,
22	DVPACK_RESERVED_RESERVED_E3 =	0xE3,
	DVPACK_RESERVED_RESERVED_E4 =	0xE4,
23	DVPACK_RESERVED_RESERVED_E5 =	0xE5,
	DVPACK_RESERVED_RESERVED_E6 =	0xE6,
24	DVPACK_RESERVED_RESERVED_E7 =	0xE7,
	DVPACK_RESERVED_RESERVED_E8 =	0xE8,
25	DVPACK_RESERVED_RESERVED_E9 =	0xE9,

```

1   DVPACK_RESERVED_RESERVED_EA =      0xEA,
2   DVPACK_RESERVED_RESERVED_EB =      0xEB,
3   DVPACK_RESERVED_RESERVED_EC =      0xEC,
4   DVPACK_RESERVED_RESERVED_ED =      0xED,
5   DVPACK_RESERVED_RESERVED_EE =      0xEE,
6   DVPACK_RESERVED_RESERVED_EF =      0xEF,
7   DVPACK_SOFT_MODE_MARKER_CODE =      0xF0,
8   DVPACK_SOFT_MODE_OPTION_F1 =        0xF1,
9   DVPACK_SOFT_MODE_OPTION_F2 =        0xF2,
10  DVPACK_SOFT_MODE_OPTION_F3 =        0xF3,
11  DVPACK_SOFT_MODE_OPTION_F4 =        0xF4,
12  DVPACK_SOFT_MODE_OPTION_F5 =        0xF5,
13  DVPACK_SOFT_MODE_OPTION_F6 =        0xF6,
14  DVPACK_SOFT_MODE_OPTION_F7 =        0xF7,
15  DVPACK_SOFT_MODE_OPTION_F8 =        0xF8,
16  DVPACK_SOFT_MODE_OPTION_F9 =        0xF9,
17  DVPACK_SOFT_MODE_OPTION_FA =        0xFA,
18  DVPACK_SOFT_MODE_OPTION_FB =        0xFB,
19  DVPACK_SOFT_MODE_OPTION_FC =        0xFC,
20  DVPACK_SOFT_MODE_OPTION_FD =        0xFD,
21  DVPACK_SOFT_MODE_OPTION_FE =        0xFE,
22  DVPACK_NO_INFO =                    0xFF,
23  } DVPACKID;
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

1          // PC1  A 1 1 B B B C C
2          // PC2  D D D D E E E E
3          // PC3  F F F F F F F F
4          // PC4  G G G G H H H H
5          //
6          // A : ME: Mic Error
7          // B : MULTI-BYTES: Maximum number of words to be writtein
8          in one cycle of multi-writing operation
9          // C : MEM TYPE: Memory Type
10         // D : MEM SIZE of SPACE 0:
11         // E : MEM SIZE of the LAST BLANK in SPACE1
12         // F : MEM BANK NO. of SPACE 1
13         // G : UNITS of TAPE THICKNESS
14         // H : 1/10 of TAPE THICKENSS
15         //
16         BOOL MicError;
17         // 0 : All events in this MIC do not always exist on this
18         tape
19         // 1 : All events in this MIC certainly exist on this tape
20         UINT32 MultiBytes;
21         // 0 : 4 bytes
22         // 1 : 8 bytes
23         // 2 : 16 bytes
24         // 3 - 6 : Reserved
25         // 7 : Unlimited
26         UINT32 MemoryType;
27         // 0 : EEPROM
28         // 1 : FeRAM
29         // Others = Reserved
30         UINT32 MemorySizeOfSpace0;
31         UINT32 MemorySizeOfLastBankInSpace1;
32         // 0 : 256 bytes
33         // 1 : 512 bytes
34         // 2 : 1 kbytes
35         // 3 : 2 kbytes
36         // 4 : 4 kbytes
37         // 5 : 8 kbytes
38         // 6 : 16 kbytes
39         // 7 : 32 kbytes
40         // 8 : 64 kbytes
41         // Others : reserved
42         // 0xF : No information
43         UINT32 MemoryBankNoOfSpace1;
44         UINT32 TapeThickness;
45     } DV_METADATA_CONTROL_CASSETTE_ID;
46
47     typedef struct _DV_METADATA_CONTROL_TAPE_LENGTH
48     {
49         DV_METADATA DVMetadata;

```

```

1          // Binary Pack Layout
2          // PC0 0 0 0 0 0 0 0 1
3          // PC1 A A A A A A AL 1
4          // PC2 A A A A A A A A
5          // PC3 AM A A A A A A A
6          // PC4 1 1 1 1 1 1 1 1
7          //
8          // A : Tape Length, MSB is at left of PC3 (M), LSB is at
9          right of PC1 (L).
10         //
11         UINT32 TapeLength;
12     } DV_METADATA_CONTROL_TAPE_LENGTH;
13
14     typedef struct _DV_METADATA_TEXT_HEADER
15     {
16         DV_METADATA DVMetadata;
17         // Binary Pack Layout
18         // PC0 0 0 0 0 1 0 0 0 (For CONTROL TEXT
19         HEADER)
20         // PC0 0 0 0 1 1 0 0 0 (For TITLE TEXT HEADER)
21         // PC1 A A A A A A AL
22         // PC2 B B B B C C C AM
23         // PC3 D D D D D D D D
24         // PC4 E E E F F F F F
25         //
26         // A : TDP: Total number of text Data (see Figure 55 of
27         part 2)
28         // B : TEXT TYPE
29         // C : OPN: Option Number
30         // D : TEXT CODE: TEXT CODE designates the character set.
31         // E : AREA NO.: Area number indicates in which area on
32         the tape this topic is stored.
33         // F : TOPIC TAG
34         //
35         UINT32 TotalTextData;
36         UINT32 TextType;
37         // 0 : Name
38         // 1 : Memo
39         // 2 : Station
40         // 3 : Model
41         // 6 : Operator
42         // 7 : Subtitle
43         // 8 : Outline
44         // 9 : Full Screen
45         // C : One byte coded font
46         // D : Two byte coded font
47         // E : Graphic
48         // F : No Information
49         // Others : Reserved
50         UINT32 OptionNumber;

```

```

1      UINT32 TextCode;
      // (See IEC 61834-4 for CONTROL TEXT header pack)
2      UINT32 AreaNumber;
      UINT32 TopicTag;
3      UINT32 cbTextPacks;
      [size_is(cbTextPacks)] BYTE    pTextPacks[];
      // text Pack Layout -- Each text pack has this layout
4      // PC0 0 0 0 0 1 0 0 1    (For CONTROL TEXT)
      // PC0 0 0 0 1 1 0 0 1    (For TITLE TEXT)
5      // PC1 ? ? ? ? ? ? ? ?
      // PC2 ? ? ? ? ? ? ? ?
6      // PC3 ? ? ? ? ? ? ? ?
      // PC4 ? ? ? ? ? ? ? ?
7      // This pack contains font data, graphic data, or text
data
8      // according to the TEXT TYPE designated in the associated
TEXT HEADER pack
9
10     } DV_METADATA_TEXT_HEADER;
11
12     typedef struct _DV_METADATA_TAG
13     {
14         DV_METADATA DVMetadata;
15         // Binary Pack Layout
16         // PC0 0 0 0 0 1 0 1 1    (For CONTROL TAG)
17         // PC1 A A A A A A AL B
18         // PC2 A A A A A A A A
19         // PC3 AM A A A A A A A
20         // PC4 C D E 1 F F F F
21         //
22         // A : Absolute Track Numnber
23         // B : Blank Flag
24         // C : Text Flag
25         // D : Temporary True
26         // E : Hold Flag
27         // F : Tag ID
28         UINT32 AbsoluteTrackNumber;
29         BOOL BlankFlag;
30         // 1 : Discontinuity exists before this absolute track
31         // 0 : Discontinuity does not exist before this absolute
32         track number
33         BOOL TextFlag;
34         // 0 : Text information exists
35         // 1 : No text information exists
36         BOOL TemporaryTrue;
37         // This flag is only valid for MIC
38         // 0 : This event data in MIC is not always valid
39         // 1 : This event data in MIC is valid
40         BOOL HoldFlag;

```



```

1      // 0 : Hold the absolute track number after playback or
recording
2      // 1 : Renew the absolute track number after playback or
recording
3      UINT32 TagId;
4      } DV_METADATA_TAG;
5
6      typedef struct _DV_METADATA_TITLE_TIME_CODE
7      {
8          DV_METADATA DVMetadata;
9          // Binary Pack Layout
10         // PC0 0 0 0 1 0 0 1 1
11         // PC1 A 1 B B C C C C
12         // PC2 1 D D D E E E E
13         // PC3 1 F F F G G G G
14         // PC4 1 1 H H I I I I
15         //
16         // A : Blank Flag
17         // B : Tens of Frames
18         // C : Units of Frames
19         // D : Tens of Seconds
20         // E : Units of Seconds
21         // F : Tens of Minutes
22         // G : Units of Minutes
23         // H : Tens of Hours
24         // I : Units of Hours
25         BOOL Blank;
26         // 0 : Discontinuity exists before the absolute track
27         // 1 : Discontinuity does not exist before the absolute
28         track
29         UINT32 Frame;
30         UINT32 Second;
31         UINT32 Minute;
32         UINT32 Hour;
33     } DV_METADATA_TITLE_TIME_CODE;
34
35     typedef struct _DV_METADATA_AUX_BINARY_GROUP
36     {
37         DV_METADATA DVMetadata;
38         DV_AUDIOBLOCK_ID DVAudioBlockId;
39         // Binary Pack Layout
40         // PC0 0 0 0 1 0 1 0 0 (For TITLE BINARY GROUP)
41         // PC1 A A A A B B B B
42         // PC2 C C C C D D D D
43         // PC3 E E E E F F F F
44         // PC4 G G G G H H H H
45         //
46         // A : Binary Group 2

```

```

1          // B : Binary Group 1
2          // C : Binary Group 4
3          // D : Binary Group 3
4          // E : Binary Group 6
5          // F : Binary Group 5
6          // G : Binary Group 8
7          // H : Binary Group 7
8          UINT32 BinaryGroup1;
9          UINT32 BinaryGroup2;
10         UINT32 BinaryGroup3;
11         UINT32 BinaryGroup4;
12         UINT32 BinaryGroup5;
13         UINT32 BinaryGroup6;
14         UINT32 BinaryGroup7;
15         UINT32 BinaryGroup8;
16     } DV_METADATA_AAUX_BINARY_GROUP;
17
18     typedef struct _DV_METADATA_BINARY_GROUP
19     {
20         DV_METADATA DVMetadata;
21         // Binary Pack Layout
22         // PC0 0 0 0 1 0 1 0 0 (For TITLE BINARY GROUP)
23         // PC1 A A A A B B B B
24         // PC2 C C C C D D D D
25         // PC3 E E E E F F F F
26         // PC4 G G G G H H H H
27         //
28         // A : Binary Group 2
29         // B : Binary Group 1
30         // C : Binary Group 4
31         // D : Binary Group 3
32         // E : Binary Group 6
33         // F : Binary Group 5
34         // G : Binary Group 8
35         // H : Binary Group 7
36         UINT32 BinaryGroup1;
37         UINT32 BinaryGroup2;
38         UINT32 BinaryGroup3;
39         UINT32 BinaryGroup4;
40         UINT32 BinaryGroup5;
41         UINT32 BinaryGroup6;
42         UINT32 BinaryGroup7;
43         UINT32 BinaryGroup8;
44     } DV_METADATA_BINARY_GROUP;
45
46     typedef struct _DV_METADATA_PROGRAM_REC_DTIME
47     {
48         DV_METADATA DVMetadata;
49         // Binary Pack Layout
50         // PC0 0 1 0 0 0 0 1 0

```

```

1          // PC1 A A BM B B B B BL
2          // PC2 CM C CL DM D D D DL
3          // PC3 EM E E FM F F F FL
4          // PC4 E E E EL GM G G GL
5          //
6          // A : Recording Mode
7          // B : Minutes
8          // C : Week
9          // D : Hours
10         // E : Year
11         // F : Day
12         // G : Month
13         UINT32 RecordingMode;
14         // 0 : Video
15         // 1 : Audio
16         // 2 : Audio Video
17         // 3 : Duplicate
18         UINT32 Minutes;
19         // 3F : No information
20         UINT32 Hours;
21         // 1F : No information
22         UINT32 Day;
23         UINT32 Month;
24         UINT32 Year;
25         // Last two digits of Year
26         UINT32 WeekDay;
27         // 0 : Sunday
28         // 1 : Monday
29         // 2 : Tuesday
30         // 3 : Wednesday
31         // 4 : Thursday
32         // 5 : Friday
33         // 6 : Saturday
34         // 7 : No Information
35     } DV_METADATA_PROGRAM_REC_DTIME;
36
37     typedef struct _DV_METADATA_AUX_SOURCE
38     {
39         DV_METADATA DVMetadata;
40         DV_AUDIOBLOCK_ID DVAudioBlockId;
41
42         // Binary Pack Layout
43         // PC0 0 1 0 1 0 0 0 0
44         // PC1 A 1 B B B B B B
45         // PC2 C D D E F F F F
46         // PC3 1 G H I I I I I
47         // PC4 J K L L L M M M
48         //
49         // A : Locked Flag
50         // B : Audio Frame Size

```

```

1      // C : Stereo Mode
2      // D : Audio Channels per audio block
3      // E : Pair Bit
4      // F : Audio Mode
5      // G : Multi-Language Flag
6      // H : 50/60
7      // I : System Type
8      // J : Emphasis Flag
9      // K : Time Constatn of Emphasis
10     // L : Sampling Frequency
11     // M : Quantization
12     BOOL LockedFlag;
13     // 0 : Locked Mode
14     // 1 : Unlocked Mode
15     UINT32 AudioFrameSize;
16     BOOL StereoMode;
17     // 0 : Multi-Stereo audio
18     // 1 : Lumped Audio
19     UINT32 Channel;
20     // 0 : One channel per audio block
21     // 1 : Two channels per audio block
22     // Others : Reserved
23     BOOL PairBit;
24     // 0 : One pair of channels
25     // 1 : Independent channel
26     UINT32 AudioMode;
27     // The interpretation of auido mode depends on the Stereo
28     Mode,
29     // the channel, and the audio block in question. See
30     section
31     // 8.1 of IEC 61834-4.
32     BOOL MultiLanguage;
33     // 0 : Recorded in Multi-Language
34     // 1 : Not recorded in Multi-Language
35     BOOL FiftySixty;
36     // 0 : 60 Field System (NTSC)
37     // 1 : 50 Field System (PAL)
38     UINT32 SystemType;
39     // Defines system type of video signal in combination with
40     50/60 flag
41     // See section 8.1 of IEC 61834-4
42     BOOL Emphasis;
43     // 0 : Emphasis on
44     // 1 : Emphasis off
45     BOOL TimeConstant;
46     // 1 : 50.15 micro-seconds
47     // 0 : Reserved
48     UINT32 SamplingFrequency;
49     UINT32 Quantization;
50     } DV_METADATA_AAUX_SOURCE;

```

```

1 typedef struct _DV_METADATA_AAUX_SOURCE_CONTROL
2 {
3     DV_METADATA DVMetadata;
4     DV_AUDIOBLOCK_ID DVAudioBlockId;
5
6     // Binary Pack Layout
7     // PC0 0 1 0 1 0 0 0 1
8     // PC1 A A B B C C D D
9     // PC2 E F G G G H H H
10    // PC3 I J J J J J J J
11    // PC4 1 K K K K K K K
12    //
13    // A : Copy Gneration Management System
14    // B : Input Source of Just Previous Recording
15    // C : Number of times Compressed
16    // D : Source Situation
17    // E : Record Start
18    // F : Record End
19    // G : Record Mode
20    // H : Insert Channel
21    // I : Direction Flag
22    // J : Speed
23    // K : Genre Category
24    UINT32 CopyGenerationManagementSystem;
25    // 0 : Copying permitted without restriction
26    // 1 : Not Used
27    // 2 : One generation of copies permitted
28    // 3 : No Copying Permitted
29    UINT32 InputSource;
30    // 0 : Analog input
31    // 1 : Digital input
32    // 2 : Reserved
33    // 3 : No Information
34    UINT32 Compression;
35    // 0 : Compressed once
36    // 1 : Compressed twice
37    // 2 : Compressed three times or more
38    // 3 : No Information
39    UINT32 SourceSituation;
40    // 0 : Scrambled source with audience restrictions
41    // 1 : Scrambled source wihout audience restrictions
42    // 2 : Source with audience restrictions or descrambled
43    // 3 : No Information
44    source with audience restrictions
45    // 3 : No Information
46    BOOL RecordingStart;
47    // 0 : Recording start point
48    // 1 : Not recording start point
49    BOOL RecordingEnd;
50    // 0 : Recording end point

```

```

1          // 1 : Not recording end point
2      UINT32 RecordMode;
3          // 1 : Original
4          // 3 : One channel inserted (CH1 or CH2 or CH3 or CH4)
5          // 4 : Four channels inserted (CH1 and CH2 and CH2 and
6      CH4)
7          // 5 : Two channels inserted (CH1 and CH2) or (CH3 and
8      CH4)
9          // 7 : Invalid Recording (MUTE)
10     UINT32 InsertChannel;
11         // 0 : CH1
12         // 1 : CH2
13         // 2 : CH3
14         // 3 : CH4
15         // 4 : CH1 and CH2
16         // 5 : CH3 and CH4
17         // 6 : CH1 and CH2 and CH3 and CH4
18         // 7 : No Information
19     BOOL DirectionFlag;
20         // 0 : Reverse Direction
21         // 1 : Forward Direction
22     UINT32 PlaybackSpeed;
23         // See IEC 61834-4 Section 8.2
24     UINT32 GenreCategory;
25         // See IEC 61834-4 Section 3.3
26 } DV_METADATA_AAUX_SOURCE_CONTROL;
27
28 typedef struct _DV_METADATA_AAUX_REC_DATE
29 {
30     DV_METADATA DVMetadata;
31     DV_AUDIOBLOCK_ID DVAudioBlockId;
32     // Binary Pack Layout
33     // PC0 0 1 0 1 0 0 1 0 (For AAUX REC DATE)
34     // PC1 A B C C D D D D
35     // PC2 1 1 E E F F F F
36     // PC3 G G G H I I I I
37     // PC4 J J J J K K K K
38     //
39     // A : Daylight Savings
40     // B : Thirty Minutes
41     // C : Tens of Time Zone
42     // D : Units of Time Zone
43     // E : Tens of Day
44     // F : Units of Day
45     // G : Week
46     // H : Tens of Month
47     // I : Units of Month
48     // J : Tens of Year
49     // K : Units of Year
50     BOOL DaylightSavingsTime;

```

```

1          // 0 : Daylight Savings Time
2          // 1 : Normal
3      BOOL ThirtyMinutesFlag;
4          // 0 : 30 Minutes
5          // 1 : 00 Minutes
6      UINT32 TimeZone;
7      UINT32 Day;
8      UINT32 Week;
9      UINT32 Month;
10     UINT32 LastTwoDigitsOfYear;
11 } DV_METADATA_AAUX_REC_DATE;
12
13 typedef struct _DV_METADATA_VAUX_REC_DATE
14 {
15     DV_METADATA DVMetadata;
16
17     // Binary Pack Layout
18     // PC0 0 1 1 0 0 0 1 0    (FOR VAUX REC DATE)
19     // PC1 A B C C D D D D
20     // PC2 1 1 E E F F F F
21     // PC3 G G G H I I I I
22     // PC4 J J J J K K K K
23     //
24     // A : Daylight Savings
25     // B : Thirty Minutes
26     // C : Tens of Time Zone
27     // D : Units of Time Zone
28     // E : Tens of Day
29     // F : Units of Day
30     // G : Week
31     // H : Tens of Month
32     // I : Units of Month
33     // J : Tens of Year
34     // K : Units of Year
35     BOOL DaylightSavingsTime;
36     // 0 : Daylight Savings Time
37     // 1 : Normal
38     BOOL ThirtyMinutesFlag;
39     // 0 : 30 Minutes
40     // 1 : 00 Minutes
41     UINT32 TimeZone;
42     UINT32 Day;
43     UINT32 Week;
44     UINT32 Month;
45     UINT32 LastTwoDigitsOfYear;
46 } DV_METADATA_VAUX_REC_DATE;
47
48 typedef struct _DV_METADATA_AAUX_REC_TIME
49 {
50     DV_METADATA DVMetadata;

```

```

1      DV_AUDIOBLOCK_ID DVAudioBlockId;
2
3      // Binary Pack Layout
4      // PC0 0 1 0 1 0 0 1 1      (For AAUX REC TIME)
5      // PC1 1 1 A A B B B B
6      // PC2 1 C C C D D D D
7      // PC3 1 E E E F F F F
8      // PC4 1 1 G G H H H H
9      //
10     // A : Tens of Frames
11     // B : Units of Frames
12     // C : Tens of Seconds
13     // D : Units of Seconds
14     // E : Tens of Minutes
15     // F : Unites of Minutes
16     // G : Tens of Hours
17     // H : Units of Hours
18
19     UINT32 Frame;
20     UINT32 Second;
21     UINT32 Minute;
22     UINT32 Hour;
23 } DV_METADATA_AAUX_REC_TIME;
24
25 typedef struct _DV_METADATA_VAUX_REC_TIME
{
    DV_METADATA DVMetadata;
    // Binary Pack Layout
    // PC0 0 1 1 0 0 0 1 1      (For VAUX REC TIME)
    // PC1 1 1 A A B B B B
    // PC2 1 C C C D D D D
    // PC3 1 E E E F F F F
    // PC4 1 1 G G H H H H
    //
    // A : Tens of Frames
    // B : Units of Frames
    // C : Tens of Seconds
    // D : Units of Seconds
    // E : Tens of Minutes
    // F : Unites of Minutes
    // G : Tens of Hours
    // H : Units of Hours
    UINT32 Frame;
    UINT32 Second;
    UINT32 Minute;
    UINT32 Hour;
} DV_METADATA_VAUX_REC_TIME;
typedef struct _DV_METADATA_AAUX_CLOSED_CAPTION
{

```



```

1 DV_METADATA DVMetadata;
2 DV_AUDIOBLOCK_ID DVAudioBlockId;
3
4 // Binary Pack Layout
5 // PC0 0 1 0 1 0 1 0 1 (For AAUX CLOSED CAPTION)
6 // PC1 1 1 A A A B B B
7 // PC2 1 1 C C C D D D
8 // PC3 1 1 1 1 1 1 1 1
9 // PC4 1 1 1 1 1 1 1 1
10 //
11 // A : Main Audio Language
12 // B : Main Audio Type
13 // C : Second Audio Language
14 // D : Second Audio Type
15 UINT32 MainAudioLanguage;
16 // 0 : Unknown
17 // 1 : English
18 // 2 : Spanish
19 // 3 : French
20 // 4 : German
21 // 5 : Italian
22 // 6 : Others
23 // 7 : None
24
25 UINT32 MainAudioType;
26 // 0 : Unknown
27 // 1 : Mono
28 // 2 : Simulated stereo
29 // 3 : True Stereo
30 // 4 : Stereo surround
31 // 5 : Data Srevice
32 // 6 : Others
33 // 7 : None
34
35 UINT32 SecondAudioLanguage;
36 // 0 : Unknown
37 // 1 : English
38 // 2 : Spanish
39 // 3 : French
40 // 4 : German
41 // 5 : Italian
42 // 6 : Others
43 // 7 : None
44
45 UINT32 SecondAudioType;
46 // 0 : Unknown
47 // 1 : Mono
48 // 2 : Descriptive video service
49 // 3 : Non-Program audio
50 // 4 : Special Effects
51 // 5 : Data Srevice
52 // 6 : Others
53 // 7 : None

```

```

1      } DV_METADATA_AAUX_CLOSED_CAPTION;
2
3      typedef struct _DV_METADATA_AAUX_TR
4      {
5          DV_METADATA DVMetadata;
6          DV_AUDIOBLOCK_ID DVAudioBlockId;
7
8          // Binary Pack Layout
9          // PC0 0 1 0 1 0 1 1 0 (For AAUX TR)
10         // PC1 A A A AL B B B B
11         // PC2 A A A A A A A A
12         // PC3 A A A A A A A A
13         // PC4 AM A A A A A A A
14         //
15         // A : Data
16         // B : Data type
17         UINT32 DataType;
18         UINT32 Data;
19     } DV_METADATA_AAUX_TR;
20
21     typedef struct _DV_METADATA_VAUX_TR
22     {
23         DV_METADATA DVMetadata;
24
25         // Binary Pack Layout
26         // PC0 0 1 1 0 0 1 1 1 (For VAUX TR)
27         // PC1 A A A AL B B B B
28         // PC2 A A A A A A A A
29         // PC3 A A A A A A A A
30         // PC4 AM A A A A A A A
31         //
32         // A : Data
33         // B : Data type
34         UINT32 DataType;
35         UINT32 Data;
36     } DV_METADATA_VAUX_TR;
37
38     typedef struct _DV_METADATA_VAUX_SOURCE
39     {
40         DV_METADATA DVMetadata;
41
42         // Binary Pack Layout
43         // PC0 0 1 1 0 0 0 0 0
44         // PC1 A A A A B B B B
45         // PC2 C D E E F F F F
46         // PC3 G G H I I I I I
47         // PC4 J J J J J J J J
48         //
49         // A : Tens of TV Channel
50         // B : Units of TV Channel
51         // C : B/W

```

```

1      // D : Enable Color
2      // E : Color frames identification
3      // F : Hundreds of TV Channel
4      // G : Source code
5      // H : 50/60
6      // I : Signal Type
7      // J : Tuner Category
8      UINT32 Channel;
9      BOOL BlackAndWhiteFlag;
10     // 0 : Black and White
11     // 1 : Color
12     BOOL ColorFramesEnableFlag;
13     // 0 : CLF is valid
14     // 1 : CLF is invalid
15     UINT32 ColorFramesId;
16     // For 525-60
17     // 0 : Color Frame A
18     // 1 : Color Frame B
19     //
20     // For 625-50
21     // 0 : 1st, 2nd Field
22     // 1 : 3rd, 4th Field
23     // 2 : 5th, 6th Field
24     // 3 : 7th, 8th Field
25     UINT32 SourceCode;
26     BOOL FiftySixty;
27     // 0 : 60 Field System (NTSC)
28     // 1 : 50 Field System (PAL)
29     UINT32 SType;
30     UINT32 TunerCategory;
31 } DV_METADATA_VAUX_SOURCE;

typedef struct _DV_METADATA_VAUX_SOURCE_CONTROL
{
    DV_METADATA DVMetadata;
    // Binary Pack Layout
    // PC0 0 1 1 0 0 0 0 1
    // PC1 A A B B C C D D
    // PC2 E 1 F F 1 G G G
    // PC3 H I J K L M N N
    // PC4 1 0 0 0 0 0 0 0
    //
    // A : Copy Generation Management System
    // B : Input source of just previous recording
    // C : The number of times of compression
    // D : Source and recorded situation
    // E : Recording start point
    // F : Record Mode
    // G : Display Select Mode
    // H : Frame/Field Flag

```

```

1          // I : First/Second Flag
2          // J : Frame Change Flag
3          // K : Interlace Flag
4          // L : Still-Field picture Flag
5          // M : Still Camera Picture Flag
6          // N : Broadcast System
7          // O : Genre Category
8          UINT32 CopyGenerationManagementSystem;
9          // 0 : Copying permitted without restriction
10         // 1 : Not used
11         // 2 : One generation of copying permitted
12         // 3 : No copying permitted
13         UINT32 JustPreviousInput;
14         // 0 : Analog
15         // 1 : Digital
16         // 2 : Reserved
17         // 3 : No Information
18         UINT32 Compression;
19         // 0 : Compresssion once
20         // 1 : Compression twice
21         // 2 : Compression three times or more
22         // 3 : No Information
23         UINT32 SourceSituation;
24         // 0 : Scrambled source with audience restrictions and
25         recorded without descrambling
26         // 1 : Scrambled source without audience restrictions and
27         recoreded wihtout descrambling
28         // 2 : Source with audience restrictions or descrambled
29         source with audience restrictions
30         // 3 : No Information
31         BOOL RecordStart;
32         // 0 : Recording start point
33         // 1 : Not recording start point
34         UINT32 RecordMode;
35         // 0 : Original
36         // 1 : Reserved
37         // 2 : Insert
38         // 3 : Invalid Recording
39         UINT32 DisplaySelect;
40         BOOL FrameField;
41         // 0 : only one of two fields is output twice
42         // 1 : both fields are output in order
43         BOOL FirstSecond;
44         // 0 : Field 2 is output
45         // 1 : Field 2 is output
46         BOOL FrameChange;
47         // 0 : Same Picture as the immediate previous frame
48         // 1 : Different Picture from the immediate previous
49         frame
50         BOOL Interlace;

```

```

1          // 0 : Non-Interlaced
2          // 1 : Interlaced or unrecognized
3      BOOL StillField;
4          // 0 : The time difference between the fields is
approximately 0s
5          // 1 : The time difference between the fiels is
approximately 1.001/60 s or
6          // 1/50 s
7      BOOL StillCamera;
8          // 0 : Still camera picture
          // 1 : Not Still Camera Picture
9      UINT32 BroadcastSystem;
10     UINT32 GenreCategory;
11 } DV_METADATA_VAUX_SOURCE_CONTROL;
12
13 typedef struct _DV_METADATA_VAUX_CLOSED_CAPTION
14 {
15     DV_METADATA DVMetadata;
16     // Binary Pack Layout
17     // PC0 0 1 1 0 0 1.0 1 (For VAUX CLOSED CAPTION)
18     // PC1 A A A A A A A A
19     // PC2 B B B B B B B B
20     // PC3 C C C C C C C C
21     // PC4 D D D D D D D D
22     //
23     // A : 1st FIELD Line 21 1st BYTE
24     // B : 1st FIELD Line 21 2nd BYTE
25     // C : 2nd FIELD Line 21 1st BYTE
26     // D : 2nd FILED Line 21 2nd BYTE
27
28     UINT32 FirstFieldFirstByte;
29     UINT32 FirstFieldSecondByte;
30     UINT32 SecondFieldFirstByte;
31     UINT32 SecondFieldSecondByte;
32 } DV_METADATA_VAUX_CLOSED_CAPTION;
33
34 typedef struct _DV_METADATA_CAMERA_CONSUMER_CAMERA_1
35 {
36     DV_METADATA DVMetadata;
37     // Binary Pack Layout
38     // PC0 0 1 1 1 0 0 0 0
39     // PC1 1 1 A A A A A A
40     // PC2 B B B B B C C C
41     // PC3 D D D E E E E E
42     // PC4 F G G G G G G G
43     //
44     // A : Iris
45     // B : Automatic Exposure Mode

```

```

1          // C : Automatic Gain Control
2          // D : White Balance Mode
3          // E : White Balance
4          // F : Focus Mode
5          // G : Focus
6          UINT32 Iris;
7          // Position in terms of F number
8          // 0 - 60 : IP where iris position =  $2^{(IP / 8)}$ 
9          // 61 : Under F1.0
10         // 62 : Close
11         // 63 : No Information
12         UINT32 AEMode;
13         // 0 : Full Automatic
14         // 1 : Gain priority mode
15         // 2 : Shutter priority mode
16         // 3 : Iris priority mode
17         // 4 : Manual
18         // 15 : No Information
19         // Others : Reserved
20         UINT32 AGC;
21         // 0 - 13 : G
22         // 15 : No Information
23         UINT32 WBMode;
24         // 0 : Automatic
25         // 1 : Hold
26         // 2 : one-push
27         // 3 : preset
28         // 7 : No Information
29         // Others : Reserved
30         UINT32 WhiteBalance;
31         // 0 : Candle
32         // 1 : Incandescent lamp
33         // 2 : Florescent lamp of low color temperature
34         // 3 : Flourescent lamp of high color tempertaure
35         // 4 : Sunlight
36         // 5 : Cloudiness
37         // 6 : Others
38         // 31 : No Information
39         // Others : Reserved
40         BOOL FocusMode;
41         // 0 : Automatic Focus
42         // 1 : Manual Focus
43         UINT32 FocusPosition;
44         // 0 - 126 : Focus Postion =  $M \times 10^L$ 
45         // where M is most significant 5 bits of focus
46         // and L is least significant 2 bits of focus
47         // 127 : No Information
48     } DV_METADATA_CAMERA_CONSUMER_CAMERA_1;
49
50     typedef struct _DV_METADATA_CAMERA_CONSUMER_CAMERA_2

```

```

1      {
2          DV_METADATA DVMetadata;
3              // Binary Pack Layout
4              // PC0 0 1 1 1 0 0 0 1
5              // PC1 1 1 A B B B B B
6              // PC2 C D E E E E E E
7              // PC3 F F F F F F F F
8              // PC4 G H H H I I I I
9              //
10             // A : Vertical Panning Direction
11             // B : Vertical Panning Speed
12             // C : Image Stabilizer
13             // D : Horizontal Panning Direction
14             // E : Horizontal Panning Speed
15             // F : Focal Length
16             // G : Zoom Enable Flag
17             // H : Units of E-Zoom
18             // I : 1/10 of E-Zoom
19             BOOL VerticalPanningDirection;
20                 // 0 : Same direction as the vertical scanning
21                 // 1 : Opposite direction as the vertical scanning
22             UINT32 VerticalPanningSpeed;
23                 // 0 - 29 : Panning Speed
24                 // 30 : More than 29 lines per field
25                 // 31 : No Information
26             BOOL ImageStabilizer;
27                 // 0 : On
28                 // 1 : Off
29             BOOL HorizontalPanningDirection;
30                 // 0 : Same direction as horizontal scanning
31                 // 1 : Opposite direction as horizontal scanning
32             UINT32 HorizontalPanningSpeed;
33                 // 0 - 30 : Panning Speed
34                 // 62 : More than 122 pixels per field
35                 // 63 : No Information
36             UINT32 FocalLength;
37                 // 0 - 254 : Focal Length
38                 // 255 : No Informtion
39             BOOL ZoomEnable;
40                 // 0 : Electronic Zoom ON
41                 // 1 : Electronic Zoom Off
42             UINT32 ElectricZoom;
43                 // 0 - 79 : 0.0 - 7.9 units of electric zoom
44                 // 126 : More than 8 times
45                 // 127 : No Information
46         } DV_METADATA_CAMERA_CONSUMER_CAMERA_2;
47
48         typedef struct _DV_METADATA_CAMERA_SHUTTER
49         {
50             DV_METADATA DVMetadata;

```

```

1      // Binary Pack Layout
2      //
3      // For Consumer Use
4      //
5      // PC0 0  1  1  1  1  1  1  1
6      // PC1 1  1  1  1  1  1  1  1
7      // PC2 1  1  1  1  1  1  1  1
8      // PC3 A  A  A  A  A  A  A  AL
9      // PC4 1  AM A  A  A  A  A  A
10     //
11     // For Professional Use
12     //
13     // PC0 0  1  1  1  1  1  1  1
14     // PC1 B  B  B  B  B  B  B  B
15     // PC2 C  C  C  C  C  C  C  C
16     // PC3 1  1  1  1  1  1  1  1
17     // PC4 1  1  1  1  1  1  1  1
18     //
19     // A : Shutter Speed
20     // B : Upper Line Shutter Speed
21     // C : Lower Line Shutter Speed
22     //
23     UINT32 ShutterSpeed;
24     // 0x0000 - 0x7FFE : Shutter Speed
25     // 0x7FFF : No Information
26     UINT32 UpperLineSpeed;
27     // 0 - 254 : Shutter Speed
28     // 255 : No Information
29     UINT32 LowerLineSpeed;
30 } DV_METADATA_CAMERA_SHUTTER;

```

## **Conclusion**

Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.